2006

The Complete MLSK Model - incorporation of lattice operations and XML implementation

Ruchi Dhingra
Iowa State University

Natalia Stakhanova
Iowa State University

Hua Ming
Iowa State University, minghuaisu@gmail.com

The Complete MLSK Model - incorporation of lattice operations and XML implementation

Abstract
Many multilevel security relational models have been proposed and different models offer different advantages. In this paper, we adapt and refine some of the best ideas from these models and add new ones of own to extend our Multilevel Security with Key-polyinstantiation (MLSK) relational model. MLSK now supports relational algebra and user lattice manipulations while ensuring that the soundness, completeness and security that it originally guaranteed are not compromised. We also implement MLSK in a non-relational scenario, thereby demonstrating the extensibility of the model to other environments.

Keywords
Multilevel Security, Key-polyinstantiation, LBAC operations, Non-relational Implementation

Disciplines
Information Security

The Complete MLSK Model – incorporation of lattice operations and XML implementation

Ruchi Dhingra Natalia Stakhanova Ming Hua

Department of Computer Science
Iowa State University
Ames, Iowa
ruchi, ndubrov, hming @iastate.edu

***Abstract*— Many multilevel security relational models have been proposed and different models offer different advantages. In this paper, we adapt and refine some of the best ideas from these models and add new ones of own to extend our Multilevel Security with Key-polyinstantiation (MLSK) relational model. MLSK now supports relational algebra and user lattice manipulations while ensuring that the soundness, completeness and security that it originally guaranteed are not compromised. We also implement MLSK in a non-relational scenario, thereby demonstrating the extensibility of the model to other environments.**

***Index Terms*—Multilevel Security, Key-polyinstantiation, LBAC operations, Non-relational Implementation**

I. INTRODUCTION

In multilevel security there is a hierarchy of users or user-levels[1], in which each user has its own version of information. A user can see all the information belonging to him and to users below his level. On the other hand, information belonging to a higher user, or even existence of such information or such user-levels, is hidden from lower user-levels. A model for a multilevel security database must be devoid of covert channels that can compromise of user confidentiality.

Existing multilevel secure (MLS) data models support u-polyinstantiation[2] [4, 6, 8, 12, 15, 16]. These have been defined for relational databases and have semantics very similar to SQL. They also have the potential to be implemented using SQL, although no such implementation appears to have been carried out.

It is important for a model to support key-polyinstantiation[3] because in the real world it is often the case [3] that an object varies in its key value(s) when it occurs in the beliefs of different users. Thus a unique key across beliefs limits our ability to accurately model the real world and any model that supports key-polyinstantiation is able to provide a more accurate and unambiguous representation of real world data.

Our earlier work was a first attempt at defining a model for a multi-level secure relational database that supports key-polyinstantiation and whose statements have semantics closely resembling SQL[5]

This time we focus on making this model more comprehensive and we extend it to include:

1. Relational algebra operations
2. Operations for user lattice manipulations
3. Incremental mapping of lattice operations in role based access control systems
4. A non-relational implementation using XML

Addition of relational algebra operations extends the usability of the model. Allowing for lattice manipulations frees the model of the current unrealistic assumption of a fixed user set with an unchanging hierarchy and with our non-relational implementation we demonstrate that users can utilize this model and capitalize on its benefits even in a non-relational environment. This makes our model the first in its class to support both relational and non-relational implementation preferences.

[1] We use both terms user and user-level interchangeably in this paper when the meaning is apparent from the context.

[2] Under u-polyinstantiation it is assumed that a real world object has the same key under beliefs of all users that can access the object, although non-key values may vary.

[3] Key-polyinstantiation allows key as well as non-key attributes to vary across user beliefs.

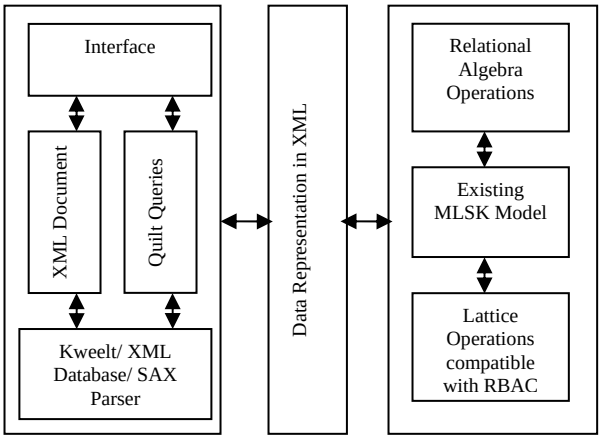

Figure 1: Scope Overview

Lattice-based access control models such as ours can be implemented in a role-based environment by generating every state of the role based model from the corresponding state of the lattice [13]. We attempted to define incremental operations for a role based model corresponding to our lattice manipulation operations, but found that any such incremental attempt cannot guarantee security and is hence infeasible.

In the following section we summarize existing research. In Section 3, we extend our MLS-K model to include relational algebra operations. In Section 4 we define lattice manipulation operations and explain why an incremental effort to modify the corresponding role-based system is infeasible. Section 5 provides an overview of the implementation[4] and describes the schema manipulation required for XML to support the model. In Section 6 we evaluate the performance of our implementation. Then we conclude with a summary of our model and areas for future work and cite references.

II. RELATED WORK

Most of the existing MLS data models support only u-polyinstantiation [4, 6, 8, 12, 15, 16]. These have been defined for relational databases and have semantics similar to SQL. They support various integrity constraints and ensure enforcement of the 'read-down and write up' policy. They also have the potential to be implemented in SQL, although no such implementation appears to have been carried out.

The only known model that supports key-polyinstantiation was proposed by [2, 3]. This model was defined for temporal and spatial databases but it also supports the relational model and it was the first work to introduce the concept of anchors. But work on this model largely remains incomplete – no operational semantics or implementation has been specified.

The first work to support key-polyinstantiation was our unpublished report on the MLSK model. In MLSK, as in any multilevel security environment, the user-levels form a fixed hierarchy where every user has its own object space. We assume that each object is uniquely identified by its key values in a given object space; although an object can have different keys when viewed in the object space of different users.

Information available to a user consists of the following:

1. The object space which is the user's belief about which objects exist in the real world
2. Property values of objects in the user's object space
3. Knowledge of which object in user's object space is known to a lower user, possibly with different identity (key) and attribute values

The formal semantics of the MLS-K data model are:

1. The structure of the model can be represented in a form of the security hierarchy of users where user-levels are partially ordered in lattice
2. Each user is assigned a *security classification* (or classification) which defines what data is visible to this user[5]
3. The relation schema is: R(OC,OO, BC,BO, A_1, C_1, A_2, C_2, …, A_k, C_k) where
   - OC- owner classification is the security classification of the tuple owner
   - OO- owner object is the object key (could be multiple attributes) as believed by the tuple owner.
   - BC- belief classification is the security classification of the relation owner
   - BO-belief object is the object key (could be multiple attributes) as believed by the relation owner
   - A_i –data attribute over domain D_i
   - C_i –classification attribute for A_i. The domain of C_i is specified by a set $\{L_i..H_i\}$ containing all security classifications ranging from Li up to Hi ($L_i < H_i \leq$ OC)

Its interpretation is:

1. Objects are believed by δ-user[6] to exist if t[OC,OO] = t[BC,BO]
2. Objects are accepted if t[OC,OO] ≠ t[BC,BO]. We say δ-user believes that an object OO is known to BC-user as object BO. And we say that the data (attributes) of object BO is accepted by δ-user
3. Objects are not accepted by a user if they are neither believed nor accepted

A user's belief comprises of all objects that have been believed and accepted by the user.

III. THE ENHANCED MLSK MODEL

We define for our model the 6 basic operations available in relational algebra:

1. Selection

[4] This is a basic implementation to prove the feasibility of implementing the relational model in a non-relational scenario. No attempt has been made to optimize performance.

[5] We use the user-level as the security classification although this is not mandatory.

[6] Note that δ is not a user in our hierarchy and when we use this notation we mean that this property is valid for the current user who could be at any level in the hierarchy i.e. δ can be μ, α, β or λ as needed.

2. Projection
3. Union
4. Intersection
5. Difference
6. Cross-product
7. Joins

For all operations we use the following:

1. R_1, R_2 , R_3, R_4 –relations on which the operations are performed
2. R - the resulting relation
3. t_1, t_2 t_3, t_4 – tuples of relations R_1, R_2, R_3, R_4, respectively
4. t - tuple of relation R
5. R = σ oc<= δ (R_1 op R_2), where op is any relational operation performed by a δ-user

Selection (σ) and Projection (π) operators are defined exactly as in classical relational algebra. Selection allows extracting rows from the relation while projection extracts columns.

To perform the Union, Intersection and Difference relations, R_1 and R_2 have to be union-compatible i.e. they have the same number of attributes and corresponding attributes (from left to right) have the same domain. The resulting relation R has schema identical to schema of R_1 and it inherits attribute names from R_1.

To illustrate these operations for user α we will consider the following relation instances:

Table 1- R_1

OC	OO	BC	BO	Age	C_1
α	Tom	λ	Hari	35	α
λ	Ron	λ	Ron	54	λ
λ	Hari	λ	Hari	39	λ

Table 3-R_2

OC	OO	BC	BO	Age	C_1
α	Tom	α	Tom	35	α
α	Tom	λ	Hari	39	λ
λ	Hari	λ	Hari	39	λ

Table 4-R_3

OC	OO	BC	BO	Salary	C_1
α	Tom	λ	Hari	80	λ
λ	Ron	λ	Ron	54	λ
λ	Hari	λ	Hari	80	λ

Table 5-R_4

OC	OO	BC	BO	Age	C_1
α	Tom	α	Tom	35	α
α	Tom	λ	Hari	39	λ
λ	Ron	λ	Ron	40	λ
λ	Hari	λ	Hari	39	λ

Table 6- $\sigma_{oc<=\alpha}(R_1 \cup R_2)$

OC	OO	BC	BO	Age	C_1
α	Tom	λ	Hari	35	α
λ	Ron	λ	Ron	54	λ
α	Tom	α	Tom	35	α
λ	Hari	λ	Hari	39	λ

Union of R_1, R_2 denoted $R_1 \cup R_2$ is the set of tuples that are in R_1, R_2 or both i.e. $\forall\ t_1 \in R_1$, $t_2 \in R_2\ \exists\ t \in R$ such that $t_1=t \vee t_2=t$.

Intersection of R_1, R_2 denoted $R_1 \cap R_2$ is the set of tuples that are in both R_1 and R_2 i.e. $\forall\ t_1 \in R_1 \wedge t_2 \in R_2$ such that $t_1= t_2$, $\exists\ t \in R$ such that $t = t_2$.

Table 7- $\sigma_{oc<=\alpha}$ ($R1 \cap R2$)

OC	OO	BC	BO	Age	C_1
α	Tom	λ	Hari	39	λ
λ	Hari	λ	Hari	39	λ

Difference of R_1, R_2 denoted R_1 - R_2 is the set of tuples that are in R_1 but not in R_2 i.e. $\forall\ t_1 \in R_1$, such that $t_1 \neq t_2$ for any $t_2 \in R_2$, $\exists\ t \in R$ such that $t = t_1$

Table 8- $\sigma_{oc<=\alpha}$ (R1- R2)

OC	OO	BC	BO	Age	C_1
λ	Ron	λ	Ron	54	λ

Cross-product of R_1, R_2, denoted R_1 x R_2 is the relation whose schema contains all attributes of R_1 followed by attributes of R_2. The resulting relation R contains one tuple t which is a concatenation of t_1 and t_2, for each pair of tuples t_1 and t_2.
Thus for user λ, R_3 X R_4 results in:

Table 9- $\sigma_{oc<=\lambda}(R_3 \text{ x } R_4)$

OC	OO	BC	BO	Age	C_1	OC	OO	BC	BO	Salary	C_1
λ	Ron	λ	Ron	40	λ	λ	Ron	λ	Ron	54	λ
λ	Ron	λ	Ron	40	λ	λ	Hari	λ	Hari	80	λ
λ	Hari	λ	Hari	39	λ	λ	Ron	λ	Ron	54	λ
λ	Hari	λ	Hari	39	λ	λ	Hari	λ	Hari	80	λ

The join operation is the most commonly used operation to combine together two relations. We apply the following steps for join implementation:

1. $\sigma_{oc<=\delta \wedge t1[OC,OO,BC,BO]=t2[OC,OO,BC,BO]}(R_1 \text{ x } R_2)$[7]
2. Predicate, if exists, is applied

Natural join, denoted $R_1 \bowtie R_2$, is a join without a predicate based on the equality of key values{OC,OO,BC,BO} i.e. R= $R_1 \bowtie R_2 = \sigma_{oc<=\delta \wedge t1[OC,OO,BC,BO]=t2[OC,OO,BC,BO]}(R_1 \text{ x } R_2)$.
Natural join eliminates duplicates.

[7] The ordering of the operators is an optimization issue that has not been considered here.

Thus, $R_3 \bowtie R_4$ results in:

Table 10- R_3 R_4

OC	OO	BC	BO	Salary	C_1	Age	C_2
α	Tom	λ	Hari	80	λ	39	λ
λ	Ron	λ	Ron	54	λ	40	λ
λ	Hari	λ	Hari	80	λ	39	λ

Thetajoin, denoted $R_1 \Theta_F R_2$, is a join of R_1 and R_2 satisfying predicate F i.e. $R_1 \Theta_F R_2 = \sigma_F (R_1 \times R_2) = \sigma_{F \wedge oc<=\delta \wedge t1[OC,OO,BC,BO]=t2[OC,OO,BC,BO]} (R_1 \times R_2)$. The predicate F can include any attribute from any relation or any comparison or boolean operators. Thus, $R_3 \Theta_{R3.oc \neq R4.c1} R_4$ results in:

Table 11- $R_3 \Theta_{R3.oc \neq R4.c1} R_4$

OC	OO	BC	BO	Salary	C_1	Age	C_2
α	Tom	λ	Hari	80	λ	39	λ

In outer join tuples from a relation that do not have matching tuples in the second relation are also included in the result. Missing values A_i, C_i are set to null. There are 3 kinds of outer join:

1. Right outer join, denoted $R_1 \bowtie R_2$, includes all tuples from R_2 with null values for tuples that do not exist in relation R_1.

$R = \sigma_{oc<=\delta \wedge t1[OC,OO,BC,BO]=t2[OC,OO,BC,BO]} (R_1 \times R_2) \cup \sigma_{oc<=\delta} (R_2)$

2. Left outer join, denoted $R_1 \bowtie R_2$, includes all tuples from R_1 with null values for tuples that do not exist in relation R_2.

$R = \sigma_{oc<=\delta \wedge t1[OC,OO,BC,BO]=t2[OC,OO,BC,BO]} (R_1 \times R_2) \cup \sigma_{oc<=\delta} (R_1)$

3. Full outer join includes all tuples of both relations.

$R = \sigma_{oc<=\delta \wedge t1[OC,OO,BC,BO]=t2[OC,OO,BC,BO]} (R_1 \times R_2)$

Thus for user α, R_3 full outer join R_4 results in:

Table 12- R_3 full outer join R_4

OC	OO	BC	BO	Salary	C_1	Age	C_2
α	Tom	α	Tom	null	null	35	α
α	Tom	λ	Hari	80	λ	39	λ
λ	Ron	λ	Ron	54	λ	40	λ
λ	Hari	λ	Hari	80	λ	39	λ

Semijoin, denoted R_1 semijoin $_F$ R_2, is a join of R_1 and R_2 satisfying predicate F where the result contains only attributes of R_1. $\pi_a(R_1 \text{ semijoin }_F R_2) = \pi_a (\sigma_{F \wedge oc<=\delta \wedge t1[OC,OO,BC,BO]=t2[OC,OO,BC,BO]} (R_1 \times R_2))$, where a is the set of all attributes of relation R_1.Thus for user α, R_4 Semijoin (BC=OC) R_3 results in:

Table 13 - R_4 Semijoin (BC=OC) R_3

OC	OO	BC	BO	Age	C_1
λ	Ron	λ	Ron	40	λ
α	Tom	α	Tom	35	α
λ	Hari	λ	Hari	39	λ

IV. LATTICE OPERATIONS AND MAPPING TO ROLE BASED ACCESS CONTROL SYSTEMS

The MLSK model, like other multi-level security models, is based on the assumption of a fixed lattice of users. This assumption restricts the application of the model, since in any realistic implementation, the user set and its structure are bound to change over time.

To ensure that the model does not compromise on any security aspect and that covert channels are avoided, we restrict the lattice manipulation operations to be carried out only by an administrator. This ensures that no user has access to the lattice and hence no user is aware of the existence of any peer or superior users.

We allow the administrator to carry out 4 basic lattice manipulation operations:

1. Add User
2. Delete User
3. Add Link
4. Delete Link

Later, we prove that these 4 operations are sufficient for any manipulation that the administrator may wish to perform on the lattice.

The add user statement has the following general form:
ADD USER <OC>
AS <parent/child> OF <OC'>
where:

1. OC is the owner classification of the new user and should not already appear in the lattice as an existing user
2. OC' represents the existing user

Some key points to note are:

1. A valid value must be entered for OC'. If, not, it would be equivalent to the new user being added a new stand alone node which denotes the creation of a new lattice[8]
2. OC cannot contain the symbol * since the use of this symbol is restricted for the Delete User statement as explained later

Thus,
ADD USER μ
AS child OF αβ

changes the lattice[9]

[8] Multi-level security models do not support multiple lattices.
[9] No data is associated with these lattices since data is relevant only to the users and these lattices are visible only to the administrator.

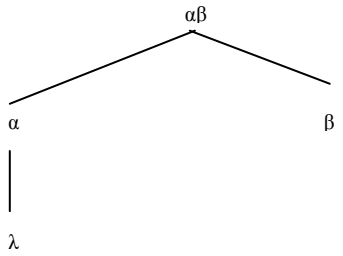

Figure 2- Initial Lattice

to:

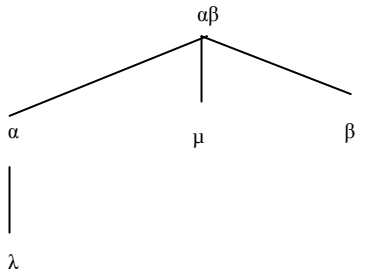

Figure 3 - Lattice After Adding User

The delete user statement has the following general form:
DELETE USER <OC>
where OC represents the user to be deleted and should be a valid entry i.e. a user who is not in the lattice cannot be deleted.
Some key points to note are:

1. When deleting a user, all children of the user (if any) should be linked directly to all parents of the user (if any) and the data should be appropriately modified
2. If deleting a user causes any user to become a stand alone user disconnected from the lattice, the action is aborted

Thus
DELETE USER α
results in:

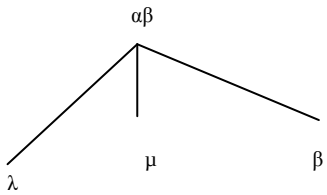

Figure 4 - Lattice After Deleting User

Note that the linkage between αβ and λ has not been implicitly deleted, instead a direct linkage has been created to avoid loss of data. Further, in the data for αβ, any belief of α previously accepted by αβ, now has C_i values α* indicating to αβ that α user has been deleted from the system.

The add link statement has the following general form:
ADD LINK FOR<OC>
AS <parent/child> OF <OC'>
where:

1. OC represents the user to be linked
2. OC' represents the parent/child user

Some key points to note are:

3. Both OC and OC' should be valid entries already existing in the lattice i.e. a user who is not in the lattice cannot be linked
4. If adding a link causes a violation of the lattice partial order, the action is aborted

Thus the statement:
ADD LINK FOR λ
AS child OF μ
results in:

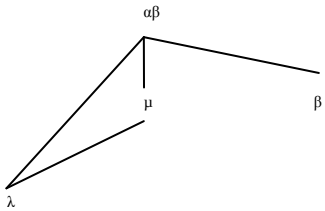

Figure 5 - Lattice After Adding Link

The delete link statement has the following general form:
DELETE LINK FOR <OC>
FROM <OC'>
where:

1. OC represents the user to be de-linked
2. OC' represents the parent/child user

Some key points to note are:

1. Both OC and OC' should be valid entries directly linked and already existing in the lattice i.e. users who are not in the lattice or not directly linked in the lattice cannot be de-linked
2. When deleting a linkage, all children of the current user (if any) should be linked to all parents of the current user (if any) and the data should be appropriately modified
3. If deleting a linkage causes any user to become a stand alone user disconnected from the lattice, the action is aborted

Thus
DELETE LINK FOR λ
FROM αβ
results in:

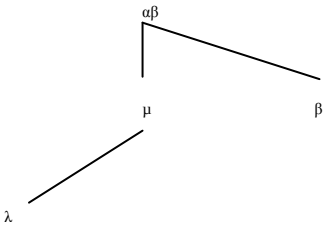

Figure 5 - Lattice After Deleting Link

These 4 operations suffice for any lattice manipulation that the administrator may wish to perform. Consider the example of the administrator wanting to move an existing user from its current position in the lattice to another position. This can be carried out by deleting the existing links between the user and its parent(s) and child(ren) and adding new links.

Thus, if the administrator wished to move μ to become parent of αβ, the following 2 operations suffice:
DELETE LINK FOR μ
FROM αβ
ADD LINK FOR μ
AS parent OF αβ

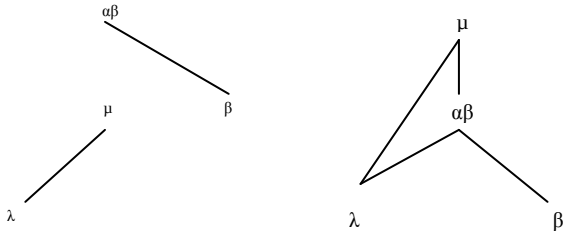

Figure 6 - Lattice Operations Equivalent to a Move

Hence no other operation is needed for lattice manipulations.

Lattice-based access control models such as ours can be implemented in a role-based environment by generating every state of the role based model from the corresponding state of the lattice [13]. We wished to define RBAC commands for each lattice operation defined by our model so that, every time a lattice L changes to a lattice we do not need to regenerate the RBAC state R' (corresponding to L') from scratch. We wanted to be able to generate R' from R (the RBAC state corresponding to L) using incremental RBAC operations.

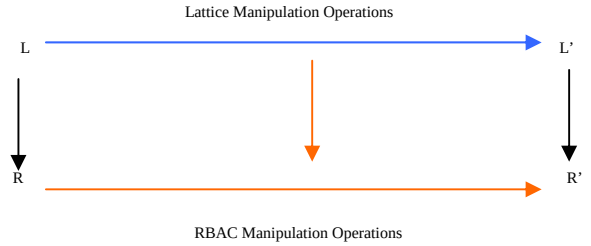

Figure 7 – Mapping Lattice Based Access Control to Role Based Access Control

However, we discovered that any lattice manipulation operation can require data propagation from the lowest to the highest level of users. This is equivalent to generating R' from L'. Further, role based systems allow users to define property inheritances and constraints with minimal restrictions [SR1993].To ensure that a lattice manipulation operation does not result in the violation of any user defined inheritance or constraint, the entire system R' needs to be validated. This endorses the fact that any incremental effort to generate R' from R does not guarantee security and to ensure security we need to generate R' from scratch using the new L' definition.

V. IMPLEMENTATION

Implementation of our model is composed of two major parts- the lattice implementation and query implementation.

Lattice implementation was developed using AT&T's grappa software package that supports lattice visualization. Four operations- add user, delete user, add link and delete link are supported in this application. In our implementation user names in lattices are specified using English alphabet to accommodate ease of input using a keyboard.

The main objective of query implementation in this demonstration software was to illustrate that our relation model can be implemented in a non-relational environment, thereby giving users greater flexibility in data management. Therefore, only simple queries were implemented. Future work will include more complicated and nested queries10.

This part of the application demonstrates MLSK queries performed on XML data. It was developed using SAX XML parser under Apache Software license, an open source XML-manipulating API called JDOM and an implanted XQuery language free software Kweelt.

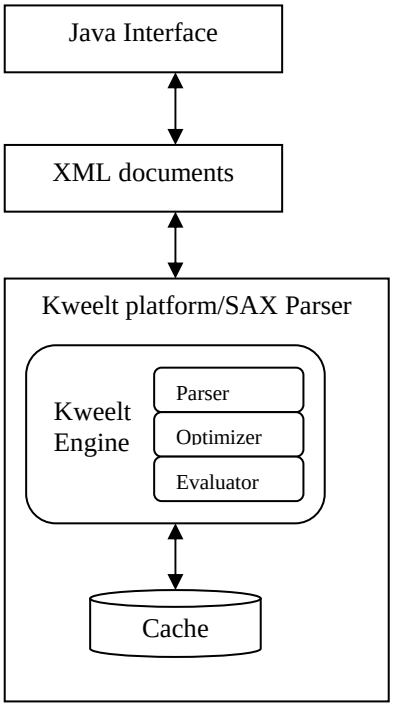

Figure 8: Application Architecture

We chose to implement our model using XML for the following reasons:

1. To demonstrate the feasibility of deploying our MLSK model in a non-relational environment
2. To avoid redundancy in data storage

[10] Sample queries are included in the Annexure

3. To provide increased flexibility in data management

In the XML schema that we used, the data is divided into two components:

1. Belief data
2. Acceptance data.

Belief data contains users' beliefs while acceptance data has 'pointers' to the data accepted by a specific user. Beliefs of individual users are specified between the tags <OC> and </OC>. Information about user objects, attributes and corresponding classification attributes is included between the tags <tuple> and </tuple> where the object is specified by the attribute OO. The id and the OO allow us to identify an object uniquely and therefore are the primary key in our XML documents. Such a nested structure allows easily identify the beginning and the end of individual objects within users beliefs and individual user beliefs within a document.

Consider the following example:

```
…<OC id="a">
  - <tuple OO="800">
      <Name>Administration</Name>
      <CName>a</CName>
      <NumAgents>403</NumAgents>
      <CNumAgents>a</CNumAgents>
      <Location>Borer Building, MT
        Complex, Orlando,
        FL</Location>
      <CLocation>v</CLocation>
    </tuple>
… </OC>
```

a is a user who believes in an object whose number is 800.

Acceptance schema indicates the data accepted by individual users. The first level tags indicate the user accepting the data. The second level contains tuple tags that provide the OO of this user belief object. The subsequent nested levels indicate user ids and their corresponding objects that are accepted by the first level user and believed to be the same as second level belief object OO.

For example, user 'a' believes that object with number 800 is known to user 'l' as object with number 7800 and known to user 'v' as object with number 2770. At the same time, since users 'l' and 'v' form a nested structure, we can say that user 'l' also believes that object with number 7800 is known to user 'v' as object with number 2770.

```
… <OC id="a">
    <tuple OO="800">
      <OC id="l">
        <tuple OO="7800">
          <OC id="v">
            <tuple OO="2770"></tuple>
          </OC>
        </tuple>
      </OC>
    </tuple>
… </OC>
```

If user 'l' does not accept user 'v's' data while user 'a' continues to accept objects of both user 'l' and 'v', then this information is reflected in the acceptance schema as:

```
… <OC id="a">
    <tuple OO="800">
      <OC id="l">
        <tuple OO="7800"></tuple>
      </OC>
      <OC id="v">
        <tuple OO="2770"></tuple>
      </OC>
    </tuple>
… </OC>
```

Since acceptance of data is not mandatory, not every user needs to be specified in acceptance schema. Further, acceptance of data does not always lead to the data borrowing. Such an acceptance schema structure allows for quick identification of all objects associated with a specific belief object of a particular user. It also provides great flexibility in object manipulation.

VI. PERFORMANCE EVALUATION

This application has been developed in Java v.1.4.1 standard edition and all experiments were run on Pentium III with 500 MH CPU and 512M of memory.

Since there is no direct parallel for this effort, there is no comprehensive benchmark available for our system evaluation. Therefore, we evaluate the scalability of our effort by measuring performance time for each operation on different data set.

Data sets based on large, medium and small lattices were created. Our small lattice comprises of 5 levels, 1-2 users per level and 1-2 objects per user. A medium lattice comprises of 10 levels, 1-3 users per level and 1-3 objects per user and a large lattice comprises of 15 levels, 2-3 users per level, 3 objects per user. For the test queries we use sets of 6, 10 and 14 queries per operation, where each set contains equal number of simple (having one selection condition) and more complex (having 2 conditions in where clause) queries.

Figure 9 shows a breakdown of the average execution time for Select statement. Execution times for queries on small and medium data sets are close unlike time for large lattice which is almost twice of time needed to execution on small lattice. This indicates that performance time increases considerably with increase in data size and a real world system will require optimization.

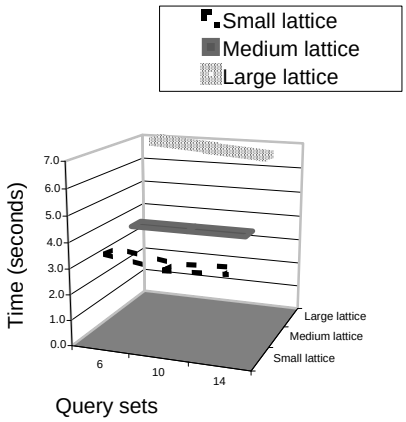

Figure 9: Execution Time for Select Statement

VII. CONLCUSION

Our work extends our MLSK model to include relational algebra, lattice manipulation operations and a non-relational implementation. These extended features eliminate some constraints of the earlier model and allow for easy adoption of the model for a real world problem.

Our next steps would be to include operations that support querying of the user lattice. The model can also be expanded to include schema definition and manipulation. Our focus will also include query optimization.

VIII. ACKNOWLEDGMENT

The authors thank Dr. Wallapak Tavanapong for her valuable comments.

REFERENCES

[1] Tomas M. Connolly and Carolyn E. Begg. Database Systems: A Practical Approach to Design, Implementation, and Management. 3d ed. Addison-Wesley, 2002

[2] Tsz Shing Cheng and Shashi K. Gadia. An Algebra for belief persistence in multilevel security databases. Technical report, Department of Computer Science, Iowa State University, September 1995.

[3] Shashi K. Gadia. Applicability of Temporal Data Models to Query Multilevel Security Databases: A Case Study. Technical report, Department of Computer Science, Iowa State University, 1998.

[4] Tsz Shing Cheng and Shashi K. Gadia. Polyinstantiation integrity in multilevel relations. Proceedings of IEEE Symposium on Research in Security and Privacy, pp 104-115, 1990.

[5] Ruchi Dhingra, Natalia Stakhanova and Wallapak Tavanapong (2002) A Multilevel Secure Relational Database Model with key-polyinstantiation. Technical Report 06-12, Department of Computer Science, Iowa State University.

[6] Sushil Jajodia and Ravi Sandhu. Toward a multilevel secure relational data model. Proceedings of ACM-SIGMOD, pp 50-59,1991.

[7] Brett McLaughlin. JAVA and XML. O'Reilly & Associates, Inc, 2000.

[8] Niki Pissinou, Kia Makki, E.K. Park. Towards a Framework for Integrating Multilevel Secure Models and Temporal Data Models. Proceedings of ACM-SIGMOD, pp 280-287, 1994.

[9] Jonathan Robie, Don Chamberlin, and Daniela Florescu. Quilt: An XML Query Language. Graphic Communication Association,. Proceedings of XML Europe, June 2000.

[10] Raghu Ramakrishnan and Johannes Gehrke. Database Management Systems. 2nd ed. McGraw Hill, 2000.

[11] Ravi Sandhu, Edward Coyne, Hal Feinstein and Charles Youman. Role-Based Access Control Models. IEEE Computer, Volume 29, Number 2, February 1996

[12] Ravi Sandhu and Fang Chen. The Multilevel Relational (MLR) Data Model. ACM Transactions on Information and System Security, Vol. 1, No. 1, November 1998, pp 93-132.

[13] Ravi Sandhu. Lattice-Based Access Control Models. IEEE Computer, Volume 26, Number 11 (Cover Article), November 1993

[14] Ravi Sandhu. Role Hierarchies and Constraints for Lattice-Based Access Control. Proceedings of the 4th European Symposium on Research in Computer Security, Rome, September 25-27, 1996

[15] Ken Smith, Marianne Winslett. Entity Modeling in the MLS Relational Model. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[16] Marianne Winslett, Kenneth smith, Xiaolei Qian. Formal Query Languages for Secure Relational Databases. ACM Transactions on Database Systems, Vol. 19, No. 4, December 1994, pp 626-662.